AF/4
IFW

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re patent application of: Docket No.: UK9-99-029

D. Renshaw

Serial No.: 09/452,927 Group Art Unit: 2122

Filed: December 2, 1999 Examiner: Chuck O. Kendall

For: **FORM DATA FILE GENERATOR**

Mail Stop Appeal Brief-Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

BRIEF OF APPELLANTS UNDER 37 C.F.R. §1.192 (c)

Sir:

Appellants have filed a timely Notice of Appeal from the Final Office Action, dated January 30, 2004, finally rejecting claims 1-15, 18-32 and 35 in this application. This Appeal Brief is being filed in triplicate pursuant to 37 C.F.R. §1.192(a).

Please charge International Business Machine Corporation's Account No. 09-0457 in the amount of $330.00 (37 C.F.R. §1.17(f)) to cover the fee for filing this Appeal Brief. Appellants believe that no extensions of time are required at this time. However, if additional extensions of time are necessary to prevent abandonment of this application, then such extensions of time are hereby petitioned under 37 C.F.R. §1.136(a), and any fees required therefor (including any additional fees for filing of the Appeal Brief) are hereby authorized to be charged, or overpayment credited, to IBM Deposit Account 09-0457.

08/19/2004 AWABI1 00000049 090457 09452927
01 FC:1402 330.00 DA

REAL PARTY IN INTEREST

The real party in interest in this appeal is International Business Machine Corporation, assignee of the entire interest in the above-identified application.

RELATED APPEALS AND INTERFERENCES

The Appellants, their legal representatives and the Assignee are not currently aware of any appeal that may directly affect or be indirectly affected by or have some bearing on the Board's decision in this appeal.

STATUS OF THE CLAIMS

Claims 1-15, 18-32 and 35 are currently pending.

1. Claims 1-4, 7, 18-21, 24 and 35 were rejected under 35 U.S.C. §103(a) by U.S. Patent No. 5,999,729 to Tabloski, Jr. et al. (Tabloski) in view of U.S. Patent No. 5,956,513 to McLain and U.S. Patent No. 6,105,119 to Kerr.

2. Claims 5, 6, 8-13, 22, 23 and 25-30 were rejected under 35 U.S.C. §103(a) over Tabloski in view of McLain, Kerr and U.S. Patent No. 6,018,627 to Iyengar et al. (Iyengar).

3. Claims 14, 15, 31 and 32 were rejected under 35 U.S.C. §103(a) over Tabloski in view of McLain, Kerr and U.S. Patent No. 6,083,276 to Davidson et al. (Davidson).

The rejections of claims 1-15, 18-32 and 35 under §103(a) are now the subject of the present appeal. The claims in issue and status of all claims are attached in Appendix "A".

STATUS OF AMENDMENTS

All prior amendments to the application have been entered.

SUMMARY OF THE INVENTION

The invention solves a current lack of programmer tools for providing definitions to a dialogue manager for creating and using forms. (p. 1, lns. 8-9). The invention also provides much needed support for syntax definition of managed dialogue forms by generating form data files which define electronic forms using visual programming methods. (Abstract). Additionally, the invention simplifies creating and debugging data files. (p. 13, lns. 1-2).

The invention achieves these advantages by streamlining programming by providing a method of using existing visual programming tools and environments, for example, IBM VisualAge for Java, (p. 17, lns. 24) to automatically generate textual definitions for forms thereby avoiding a need to make further large investments in next-generation programming tools. The graphical layout of the invention additionally means that it is relatively simple for an observer to grasp how the program fits together. (p. 17, lns. 7-9). Furthermore, all aspects of user interface or presentation and business logic are dealt with in one simple application, and the application is thus a self-contained entity or closely related to other such applications. (p. 17, lns. 9-12).

In particular, in a method, system and computer program code of the invention, data files are created using a programming development environment on a computer system. The steps of the method include building a program in a development environment to represent a data file (p. 12, lns. 1-5), and compiling the program in the development environment into a software executable. (p. 12, lns. 5-7). The executable is run to generate the data file containing definition files which are interpreted by a third party computer system for future application execution. The third party computer system comprises a dialogue management system for a computer telephony system. (p. 17, lns. 18-21).

ISSUES

1. Whether the subject matter of claims 1-4, 7, 18-21, 24 and 35 is obvious in view of the level of ordinary skill in the art under 35 U.S.C. §103(a) as evidenced by U.S. Patent No. 5,999,729 to Tabloski, Jr. et al. (Tabloski) in view of U.S. Patent No. 5,956,513 to McLain and U.S. Patent No. 6,105,119 to Kerr.

2. Whether the subject matter of claims 5, 6, 8-13, 22, 23 and 25-30 is obvious in view of the level of ordinary skill in the art under 35 U.S.C. §103(a) as evidenced by Tabloski in view of McLain, Kerr and U.S. Patent No. 6,018,627 to Iyengar et al. (Iyengar).

3. Whether the subject matter of claims 14, 15, 31 and 32 is obvious in view of the level of ordinary skill in the art under 35 U.S.C. §103(a) as evidenced by Tabloski in view of McLain, Kerr and U.S. Patent No. 6,083,276 to Davidson et al. (Davidson).

GROUPING OF CLAIMS

Claims 1-4, 6-8, 14, 15, 18-21, 24, 25 31, 32 and 35 stand or fall together. Claims 5, 9-13, 22, 23 and 26-29 stand or fall together. The reasons for these groupings are provided below.

ARGUMENT

The invention is directed to a form data files generator and relates to visual programming, and more particularly, to the generation of form data files which define electronic forms using such technology. (p. 1, lns 7-9). Unlike the combination of references applied by the Examiner, the invention provides a simple graphical means for producing a program, which, when run, creates form definitions to be used by a dialogue manager. Distinct from the references, the invention does not create an application, but rather, creates a program which creates an application. Where the applied references employ a one step process to create a program, the invention utilizes a process where an

executable program from a development process is run in order to generate a set of forms definitions from data files.

For example, in one aspect of the invention, a method, system, and computer program code is provided for creating a data file using a programming development environment on a computer system. The embodiments of the invention include building a program in the development environment to represent a data file is provided. (p. 12, lns. 1-5). These embodiments further include compiling the program in the development environment into a software executable. (p. 12, lns. 5-7). The executable generates the data file containing definition files which are interpreted by a third-party computer system for future application execution. Additionally, the third-party computer system comprises a dialog management system for a computer telephony system. (p. 13, lns. 13-15). As such, embodiments of the invention provide a method of using existing visual programming tools and environments to automatically generate textual definitions for forms. This avoids a need to make further large investments in next-generation programming tools.

Claims 1-4, 6-8, 14, 15, 18-21,
24, 25 31, 32 and 35

The Examiner asserts that Tabloski shows, in part, building a program to represent a data file, compiling it into a software executable, and running the executable to generate the data file. Appellants respectfully do not agree with the Examiner's interpretation of Tabloski.

Tabloski shows a code generated by a program composition module where the code comprises a high-level programming language which is then compiled by a compiler to be executed by a parallel computer. (col. 20, lns. 20-35). Typically, the high-level language is C++. (col. 20, ln. 25). This is in contrast to claim 1, where an executable is run to generate a data file containing definition files which is then run by another computer. As such, the executable of claim 1 generates a data file to be run,; whereas, the executable program of Tabloski is itself executed by another computer.

The Examiner is of the opinion, though, that Tabloski shows at col. 20, lines 30-35

> The compiler 34, compiles the executable program, using a runtime library 36. Thereafter, the executable program 25 can be executed by the parallel computer 25 under control of the run-time system 26 ..... Here Tabloski provides support for compiling and executing code. There is no teaching in Applicant's limitations that excludes the use of a parallel system to execute the code after it's been compiled, therefore Applicant is arguing for an unclaimed merit of distinction and hence Applicant's argument is moot.

However, Appellants submit that running the executable to generate the data file containing definition files which are interpreted by a third party computer system for future application execution, as set forth in claim 1, does distinguish over Tabloski's first system running under the control of, and therefore simultaneously with a second system. Specifically, since embodiments of the invention, as set forth by claim 1, are directed to interpreting files for future application execution, it is clear that claim 1 distinguishes over the Tabloski's two systems running concurrently, as required when one system controls another.

The Examiner additionally notes that Tabloski provides support for "compiling and executing the code" and that "[t]here is no teaching in Applicant's limitations that excludes the use of a parallel system to execute code after its been compiled." (Advisory Action, p. 2, 2nd paragraph). However, as noted above, claim 1 sets forth definition files which are interpreted by a third party computer system for future application execution. Future application execution by a third party clearly distinguishes over concurrently running parallel systems, as set forth by Tabloski.

The Examiner admits that Tabloski fails to disclose data files being interpreted by a third party computer system and claims that McLain discloses this feature by citing from McLain that "ABC 123 will identify from source modules every header file, and

will retrieve these from either a project library 123 or from a third party header and shared library 115." (Advisory Action, p. 2, 3rd paragraph). However, McLain discloses reading header files which is distinct from the interpreting definition files of claim 1. Specifically the header files of McLain are header files that are read so that the McLain system can identify any embedded header files therein. (col. 8, lns. 2-4). Definition files which are interpreted by a third party computer system for future application execution, as set forth in claim 1, is distinct from simply scanning a file for another file embedded therein.

Being even more specific, the independent claims show creating a data file. However, Tabloski does not disclose generating a data file in any way. Instead, Tabloski enables data transfer from a data file to a parallel computer for processing. (col. 22 54-58). The independent claims further show building a program to represent a data file. In contrast, Tabloski simply facilitates transferring data from a source to a processor. *Id.* The independent claims further recite running an executable to create definition files to be interpreted by a 3rd party computer for future application. Tabloski shows distributing program code for executable objects among processing elements of a parallel computer. (col. 2, lns. 41-43). An execution control object controls processing of the executable objects using a dataflow processing model (col. 2, lns. 41-45). However, there is no disclosure, whatsoever, that the execution control object is also passed to the processing elements, and thus remains on the program development computer. Accordingly, the program development computer must also run in parallel during processing obviating the possibility of interpretation by a third party computer system for future application.

In any event, the Examiner acknowledges that Tabloski does not explicitly disclose that a data file containing definition is interpreted by a third-party computer system. For his feature, the Examiner cites McLain. However, regardless of whether McLain supplies this missing feature of Tabloski, McLain fails to show running the executable to generate a data file and thus fails to supply the missing feature of Tabloski as discussed above. For example, McLain shows a configuration data file ("CDF") which is a text file that is created by a user. (col. 7, lns. 48-49). As such, because the

CDF is a text file that is created by a user, McLain fails to show running an executable to generate the data file. Thus, McLain fails to cure the deficiencies of Tabloski as described above.

The Examiner further cites Kerr for disclosing use of data files (header files) in third-party systems and vendors in a dialog system. But, Kerr fails to cure the deficiencies of Tabloski and McLain, as discussed above. In particular, rather than showing running an executable to generate a data file containing definition files which are interpreted by a third-party computer system for future application execution, Kerr is directed to data transfer circuitry having source code which is configured to lead to identically located data structures, no matter what type of compiler is used. (col. 21, lns. 41-42). Furthermore, Kerr shows using the common data structure to control data flow through system memory space so that the code is processor-independent. (col. 21, lns. 62-65).

Accordingly, Kerr is directed to showing technology which may be applied at any point in a PC system, but fails to disclose or suggest running an executable to generate a data file containing definition files which are interpreted by a third-party computer system for future application execution. Thus, Kerr fails to cure the deficiencies of Tabloski and McLain.

Appellants also submit that there is no motivation to combine Tabloski, McLain and Kerr. Tabloski is directed towards simplifying development and processing of programs for parallel systems. (col. 1, lns 33-35). McLain is directed towards automating the software build control process by using original program files as input and automatically identifying all shared library files before compilation. (col. 1, lns 5-11). Nothing in McLain suggests it is compatible with parallel processing software or processors. Additionally, McLain evaluates compiling and linking processes prior to performing them to detect potential errors and conflicts. (col. 3, lns. 61-64). Tabloski does not disclose a need for pre-compilation error and conflict checking because Tabloski works with a predetermined set of components (col. 1, lns. 41-42) designed to be compatible with one another, and the execution control object automatically corrects

blocking conditions (arising when an execution object does not have data) and resumes operation when the blocking condition is corrected. (col. 2, lns. 48-53). Also, because Tabloski is directed to creating software for parallel computers, and McLain is directed to identifying header files and listing unresolved conflict errors, neither reference has a need for the dialogue management features of Kerr.

Furthermore, even if one were to attempt to combine the dialogue management system of Kerr, into Tabloski with McLain, there is absolutely no information offered by any of the reference how to make such a combination without destroying the functionality of the systems of Tabloski. For example, Tabloski only works with a predetermined set of components (col. 1, lns. 41-42). Neither McLain nor Kerr mention having components compatible with a system for developing computer programs for execution on a parallel processing system. Neither McLain nor Kerr mention how to convert their components into the required predetermined set of components necessary to function in the Tabloski system.

For the reasons discussed above, Tabloski, McLain and Kerr, in combination, do not disclose or suggest running an executable to generate a data file containing definition files which are interpreted by a third-party computer system for future application execution, as set forth in claims 1, 18 and 35. As such, the Examiner has failed to establish a *prima facie* case of obviousness over claims 1, 18 and 35 and such claims are in allowable condition. Claims 2-4, 6-8, 14, 15, 19-21, 24, 25, 31, and 32 are allowable at least for the reasons discussed above with respect to independent claims 1 and 18, from which they respectively depend.

Claims 5, 9-13

22, 23 and 26-28, 29, 30

The Examiner asserts that Iyengar shows running an executable and at least one compiled component creates a file output stream and writes its respective data file to the output screen. However, Iyengar shows that each tool employed during a development process puts information into a repository and takes information out of the repository. As

such, the system integrates tools used in different parts of the development process by passing necessary information from one tool to another. Accordingly, Iyengar fails to show that on running an executable at least one compiled component creates a file output stream and writes its respective data file information to the output stream, as set forth in claim 5.

More specifically, Iyengar is directed to a tool-independent system for application building in an object oriented development environment with data stored in a repository in OMG-compliant UML representation. Iyengar shows that output data from a developmental tool is transferred into a generic format data which is saved in a repository. (col. 5, lns. 35-38). The repository also contains all output data, application components, and information as to the relationship between the entities and objects stored in the repository. Each tool employed during the development process puts information into the repository and takes information out of the repository. (col. 4, lns. 22-24). In this way, the system integrates the tools used in different parts of the development process by passing necessary information from one tool to another. (col. 4, lns. 24-26). But, Iyengar fails to disclose or suggest running an executable to generate a data file containing definition files which are interpreted by a third-party computer system for future application execution, in contrast to the Examiner's assertion.

In fact, Iyengar does not discuss executable programs generating data files in any way because Iyengar is directed to a tool-independent system for application building. Additionally, the inputting and outputting of information to and from the repository is not done by running an executable file, and furthermore, such information is not interpreted by a third-party computer system for future application execution.

Claims 5 and 22 set forth the further distinguishing feature of wherein on running the executable, at least one compiled component creates a file output stream and writes its respective data file information to the output stream. The Examiner admits that Tabloski, McLain and Kerr fail to disclose the features of claims 5 and 22, and cites Iyengar for the missing features. Specifically, the Examiner cites Iyengar for "putting and taking output data from a repository." However, putting or taking output data from a

repository is a one step transfer process. Also, this data is not the same as at least one compiled component creating an output file stream. Accordingly, the one step data transfer process of Iyengar is distinct from the process of creating a file output stream and writing its respective data file information to the output stream, as set forth in claim 5 and 22.

Claim 23 sets forth the further distinguishing feature of wherein on running the executable, at least one compiled component causes another component to output its respective data file information into the data file. The Examiner cites to Figure 3 of Tabloski for this feature. However, Figure 3 of Tabloski actually shows a program development window which allows a developer to customize the instances of respective components and modules by providing information through dialogue boxes that are associated with respective icons which will be used later to control generation of the high level program by the program composition module. (col. 13, lns. 56-63). Accordingly, Figure 3 fails to show data files and also fails to show outputting respective data as set forth in claim 23.

Claims 9 and 26 set forth the further distinguishing feature of wherein the development components comprise a main component and a sub-component. Claims 10, 11 and 27, 28 depend from claims 9 and 26, respectively. The Examiner cites Tabloski at col. 13, ln. 65 – col. 14 ln. 10 for the features of claim 9 and 26. However, Tabloski actually shows an icon representing a file containing input data to be processed (col. 13, lns. 65-66), and a dialog box having a number of input fields (col. 14, lns. 8-9). Thus Tabloski fails to show main components and sub-components of a development component as set forth in claims 9 and 26.

Claims 12 and 29 set forth the further distinguishing feature of wherein the program is compiled by generating an executable component from each development component and linking the executable components together. Claims 13 and 30 depend from claims 12 and 30, respectively. The Examiner cites Tabloski at col. 6, lns. 18 – 23 for the features of claim 12 and 29. However, Tabloski shows that a high level language program generated by a program composition module which is represented by a high-

level language program is compiled by a compiler during which a run-time library is linked in a conventional manner to generate an executable library. Consequently, in contrast to generating an executable directly from each development component set forth in claims 12 and 29, Tabloski requires an intervening run-time library to link to the compiler to create an executable library.

Accordingly, claims 5, 6, 8-13, 22, 23 and 25-30 are allowable at least for the reasons discussed above with respect to claims 1, 18 and 35, from which they respectively depend, as well as for their added features. Applicant respectfully requests the rejection of claims 5, 6, 8-13, 22, 23, and 25 be withdrawn.

CONCLUSION

In summary, the combination of references, as suggested by the Examiner, does not teach or suggest the features of the claimed invention. Therefore, the references do not provide evidence that would support a conclusion of obviousness under 35 U.S.C. §103(a). Appellants thus respectfully submit that the rejections of claims 1-15, 18-32 and 35 are in error and reversal thereof is respectfully requested.

Respectfully submitted,

Andrew M. Calderon
Reg. No. 38,093

McGuireWoods, LLP
1750 Tysons Boulevard
Suite 1800
McLean, VA 22102-4215
Tel: 703-712-5426
Fax: 703-712-5285

APPENDIX "A"

CLAIMS

A copy of all entered claims and a status of the claims is provided below.

1. A method for creating a data file using a programming development environment on a computer system, said method comprising the steps of:

building a program in said development environment to represent said data file;

compiling the program in said development environment into a software executable; and

running the executable to generate the data file containing definition files which are interpreted by a third party computer system for future application execution, and the third party computer system comprises a dialogue management system for a computer telephony system.

2. The method of claim 1 wherein the program is built by linking a plurality of development components

3. The method of claim 2 wherein at least one component comprises characteristic data file information

4. The method of claim 3 wherein on running the executable, at least one compiled component outputs its respective data file information into the data file.

5. The method of claim 4 wherein on running the executable, at least one compiled component creates a file output stream and writes its respective data file information to the output stream.

6. The method of claim 4 wherein on running the executable, at least one compiled component causes another component to output its respective data file information into the data file.

7. The method of claim 2 wherein at least one development component comprises a graphical icon for a visual development graphical user interface.

8. The method of claim 2 wherein the development components are Java beans.

9. The method of claim 2 wherein the development components comprise a main component and a sub-component.

10. The method of claim 9 wherein the main development component represents a form.

11. The method of claim 10 wherein the sub-component represents a text field on the form.

12. The method of claim 2 wherein the program is compiled by generating an executable component from each development component and linking the executable components together.

13. The method of claim 12 wherein on running a first executable component, data file information from the first executable is output before running the next and subsequent executable components.

14. The method of claim 1, wherein the data file comprises mark-up information.

15. The method of claim 14, wherein the mark-up information comprises XML.

16-17. (Canceled)

18. A system for creating a data file using a programming development environment on a further computer system, said system comprising:

means for building a program in said development environment to represent said data file;

means for compiling the program in said development environment into a software executable; and

means for running the executable to generate the data file containing definition files which are interpreted by a third party computer system for future application execution, and the third party computer system comprises a dialogue management system for a computer telephony system.

19. The system of claim 18 wherein the program is built by linking a plurality of development components.

20. The system of claim 19 wherein at least one component comprises characteristic data file information.

21. The system of claim 20 wherein on running the executable, at least one compiled component outputs its respective data file information into the data file.

22. The system of claim 21 wherein on running the executable, at least one compiled component creates a file output stream and writes its respective data file information to the output stream.

23. The system of claim 21 wherein on running the executable, at least one compiled component causes another component to output its respective data file information into the data file.

24. The system of claim 19 wherein at least one development component comprises a graphical icon for a visual development graphical user interface.

25. The system of claim 19 wherein the development components are Java beans.

26. The system of claim 19 wherein the development components comprise a main component and a sub-component.

27. The system of claim 26 wherein the main development component represents a form.

28. The system of claim 27 wherein the sub-component represents a text field on the form.

29. The system of claim 19 wherein the program is compiled by generating an executable component from each development component and linking the executable components together.

30. The system of claim 29 wherein on running a first executable component, data file information from the first executable is output before running the next and subsequent executable components.

31. The system of claim 18, wherein the data file comprises mark-up information.

32. The system of claim 31, wherein the mark-up information comprises XML.

33-34. (Canceled)

35. A computer program product comprising computer program code stored on a computer readable storage medium for, creating a data file using a programming development environment, said computer program product comprising:

means for building a program in said development environment to represent said data file;

means for compiling the program in said development environment into a software executable; and

means for running the executable to generate the data file containing definition files which are interpreted by a third party computer system for future application execution, and the third party computer system comprises a dialogue management system for a computer telephony system.

\\COM\438395.1